УДК 004.056.53

Коломыцев М.В., Носок С.А.

АНАЛИЗ УЯЗВИМОСТЕЙ ПРОТОКОЛОВ АУТЕНТИФИКАЦИИ WEB

Современная тенденция создания WEB-приложений состоит в персонализации контента и управляемом доступе к предоставляемым сервисам. В связи с этим, повышаются требования к аутентификации клиентов WEB-сайтов. Существующие подходы к построению механизма аутентификации зачастую уязвимы к атакам. В статье рассматриваются различные ограничения и меры, обеспечение которых позволит снизить риск нарушений безопасности.

Ключевые слова: WEB-приложения, протокол, ресурс.

Современная тенденция создания WEB-приложений состоит в персонализации контента и управляемом доступе к предоставляемым сервисам. В связи с этим, повышаются требования к аутентификации клиентов WEB-сайтов. Существующие подходы к построению механизма аутентификации зачастую уязвимы к атакам. В статье рассматриваются различные ограничения присущие используемым процедурам аутентификации и меры, обеспечение которых позволит снизить риск нарушений безопасности.

Актуальность задачи безопасности систем аутентификации

Консорциум Web Application Security Consortium (http://www.webappsec.org) периодически представляет статистику уязвимостей и угроз безопасности Web-приложений. В предлагаемой ими классификации угроз отдельный класс угроз связан с атаками на используемые методы аутентификации [1]. Связаны они с особенностями используемых в WEB схем аутентификации и их потенциальными ограничениями.

Ограничивающими факторами при выборе схемы аутентификации являются:

- ограничения на сложность реализации. Используемые технологии должны поддерживаться на разных платформах и не создавать существенной нагрузки на сервер. Поэтому технологии, реализующие сложные вычисления на стороне клиента (с использованием например, Javascript, Java, ActiveX и FlashB) используются не часто. Наиболее распространенной формой обмена аутентифицирующей информацией в последовательности HTTP запросов являются *cookie,*

- восприятие пользователями. Форма диалога должна быть максимально простой, и исключать необходимость установки дополнительных компонент на компьютере клиента, либо организации процедуры аутентификации в виде длинной последовательности диалоговых окон,

- производительность. Защищенные протоколы аутентификации, криптографические преобразования вообще, существенно снижают производительность сервера. Это в частности, касается протокола SSL.

Уязвимости протоколов аутентификации в WEB

Базовая аутентификация

Это простой протокол проверки подлинности, поддерживаемый всеми браузерами [3]. Его еще иногда называют HTTP аутентификацией. Протокол работает следующим образом:

- ресурсы, для доступа к которым необходимо пройти аутентификации, помещаются в отдельный каталог, и создается файл конфигурации, описывающий местоположение базы учетных записей и набор ограничений,

- браузер отправляет HTTP запрос на доступ к защищенному ресурсу, который выглядит так:

GET /test/secure HTTP/1.0,

- серверный HTTP процесс определяет, что необходим доступ к защищенному ресурсу и считывает из файла конфигурации параметры (имя домена). Выяснив, что пользовательский запрос не содержит аутентифицируещей информации, он его отвергает, сообщая клиенту, что требуется аутентификация с помощью пароля:

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="luxor",

- браузер, получив отказ, проверяет, нет ли в кеше пароля и логина для указанной области данного сервера. Если не находит, то выводит диалоговое окно для ввода имени и пароля. Введенные пользователем данные сохраняются для дальнейшего использования,

- браузер повторяет запрос к серверу, но уже с информацией о имени и пароле пользователя. Пароль пользователя передается в зашифрованном (Base64) виде:

```
GET /test/secure HTTP/1.0
Authorization: Basic dGVzdDp0ZXN0,
```

- серверный HTTP процесс сверяет полученную информацию с той, которая хранится в базе учетных записей. Если учетная запись с таким именем не существует, или переданный пароль не совпадает с тем, который хранится в базе, сервер отвергает запрос. Получив отказ, браузер повторно выводить окно для ввода имени или пароля,

- если все проверки оказались успешными, сервер пересылает браузеру нужную HTTP страницу.

Теоретически, для доступа к каждой защищенной страничке необходимо указать имя и пароль. На практике, браузер пытается получить доступ к защищаемым страничкам, используя имя и пароль, хранимые в кеше браузера.

Недостаток протокола– пароль шифруется, но очень слабым алгоритмом (Base64).

Имя пользователя и пароль пересылаются в заголовке Authorization и могут быть перехвачены. Зашифрованный пароль несложно расшифровать, например, с помощью следующей программы на Perl:

```
#!/usr/bin/perl
# bd64.pl
# decode from base 64
use MIME::Base64;
print decode_base64($ARGV[0]);
```

Программу можно выполнить, указав перехваченный пароль как параметр:

```
C:\bd64.pl dGVzdDp0ZXN0
test:test    .
```

Другим, важным с точки зрения безопасности, обстоятельством является то, что установленное доверительное соединение может быть разорвано только путем закрытия браузера. Даже если пользователь переключил браузер на сайт, не требующий аутентификации, имя и пароль хранятся в памяти. Таким образом, оставленный без присмотра компьютер может служить средством, для беспрепятственного доступа других пользователей к защищенным ресурсам.

Аутентификация на основе хеша

Данный метод аутентификации был добавлен в протокол HTTP, для исправления основных недостатков базовой аутентификации. Во многом этот метод работает так же, как и базовая аутентификация [4]. При попытке доступа браузера к защищенному ресурсу, запрос отвергается. Но в сообщении сервера указывается, что требуется аутентификация на основе хеша и содержится значение (*nonce*), индивидуальное для каждого запроса. Это значение может формироваться, например, по значению текущего времени и ip-адреса клиента. По умолчанию, при вычислении хеша используется алгоритм MD5. Затем:

- браузер получает пароль пользователя (из диалогового окна или из памяти);
- объединяет в одну строку имя пользователя, имя области аутентификации и пароль, затем вычисляет хеш полученного значения (*хешA*);
- объединяет в одну строку запрашиваемый URL с названием метода, используемого в запросе (PUT, GET, …) и вычисляет хеш полученной строки (*хешB*);
- выполняет конкатенацию *хешA* и *хешB* со значением *nonce* и вычисляет результирующий хеш;
- полученное значение хеша отправляется серверу;
- когда сервер получает клиентский запрос, он извлекает из хранилища пароль пользователя, на его основе повторяет вычисления контрольных значений хеша.

Возможна реализация алгоритма, когда хранится не пароль пользователя, а вычисленное значение *хешA*. Процедура аутентификации считается успешной, если результат вычислений совпадают с тем, что получено от клиента.

Протокол аутентификации на основе хеша является стойким к перехвату пароля, однако, его нельзя считать полностью защищенным. Он уязвим к атакам повтора (replay) запроса. Если трафик не шифруется с помощью SSL или HTTPS, то атакующий может просматривать все запросы в текстовом виде. Он может сохранить текст запроса целиком и позднее повторить его.

Поскольку оригинальный запрос браузера находится в хеше, то атакующий может получить доступ только к определенным ресурсам. Если запрашиваемым ресурсом является статическая HTML страница, то он получит ее копию (которую, впрочем, он и так мог получить, просто перехватывая трафик между клиентом и сервером). Если же HTML страницы формируются динамически, то ресурсом является сценарий сервера, и последствия такой атаки могут быть гораздо серьезнее. Решением этой проблемы может быть хранение отправленного клиенту значения *nonce* и запрет его повторного использования.

Однако такой вариант является достаточно затратным для ресурсов сервера. Более простой (и менее надежный) способ – формирование *nonce* на основе информации, которую трудно подделать: ip-адреса клиента, текущей даты и временной метки. HTTP сервер примет запрос на аутентификацию, только если он поступил из того же ip-адреса и не является слишком старым.

Другая уязвимость протокола состоит в том, что сервер ограничен в возможности защищенного хранения паролей. Поскольку для вычисления хеша используется пароль, сервер должен иметь доступ к паролю в открытом виде, либо он должен хранить вычисленное значение *хешA*. Таким образом, данный протокол нельзя использовать, если пароли защищены с помощью необратимого шифрования. В этом смысле данный протокол более уязвим, чем протокол базовой аутентификации. Если неавторизованный пользователь получает доступ к базе паролей, он получает доступ к хешу, сформированному на основе имени пользователя и пароля. В этом случае, ему нет необходимости знать пароль, он может воспользоваться значением *хешA* для атаки данного сайта. Таким образом, безопасность базы паролей при использовании данного протокола является важнейшей задачей.

Аутентификация на основе форм

Данный протокол аутентификации пожалуй, чаще всего используется в Web-приложениях. Хотя стандартной реализации данного метода нет, его преимуществом является возможность гибкого формирования алгоритма реализации. Разработчики сами определяют детали его функционирования.

В этом протоколе для ввода имени и пароля пользователя используются HTML формы. Форма – это основной способ ввода информации на стороне клиента, для отправки ее серверу (точнее, Web-приложению) с помощью тегов FORM и INPUT.

Например, для ASP.NET способ аутентификации задается в файле web.config строками

```
<authentication mode="Forms">
</authentication>  .
```

Этот файл определяет защищаемые ресурсы, а так же содержит имена пользователей и пароли (или их хешы). Имена пользователей и пароли могут храниться и в специально организованном хранилище, например, базе данных.

Кроме того, должна присутствовать форма для ввода логина и пароля (login.aspx).

Процедура аутентификации происходит следующим образом:

- пользователь отправляет запрос на доступ к ресурсу (странице default.aspx)

```
GET /default.aspx HTTP/1.0
```

- сервер перенаправляет запрос к форме для ввода пароля

```
HTTP/1.1 302 Found
Location: /login.aspx?ReturnUrl=%2fdefault.aspx  ,
```

- в браузере пользователя отображается форма login.aspx . Она содержит поля для ввода имени пользователя и его пароля. После того, как пользователь укажет имя, пароль и нажмет кнопку «Ок», введенная информация будет отправлена серверу:

```
POST /login.aspx?ReturnUrl=%2fDefault.aspx HTTP/1.0
STATE=gibberish&txtUser=test&txtPassword=test  .
```

При использовании метода GET введенные данные пересылаются в URL, в методе POST они пересылаются в теле запроса. Всегда должен использоваться метод POST, чтобы избежать хранения аутентификаторов на стороне клиента (в истории браузера), кеширования их прокси серверами. Да и сервера приложений зачастую сохраняют информацию о URL для сбора статистики. Поскольку информация об аутентификаторах помещается в ответ браузера в открытом виде, она передаются на сервер, обычно по протоколу SSL или TLS.

Проблема безопасности HTML-форм заключается в том, что по своей природе Web-приложения являются приложениями «без предыстории» (stateless). Если для доступа к ресурсу пользователь должен быть аутентифицирован, то возникает необходимость сохранения учетных данных, введенных в поля формы, чтобы передать на сервер как часть следующей транзакции. Чтобы каким-то образом сохранить аутентифицирующую информацию, разработчики могут использовать два подхода – сохранять необходимую информацию на сервере либо на стороне клиента, причем большинство разработчиков предпочитают второй вариант, считая его более легким в реализации. Информация на стороне клиента может сохраняться в файлах *cookies* браузера, в виде значений, добавляемых к URL и в скрытых полях HTML-форм. Уязвимость последних двух вариантов очевидна [2]. В случае, если информация о текущей сессии пользователя хранится на сервере (в виде временных файлов или временных таблиц в базе данных), ссылки на эту информацию должны быть доступны на стороне клиента. Как правило, они хранятся в виде ***Id*** сессии в файлах *cookie* .

- получив логин и пароль сервер сверяет их теми, что хранятся в файле web.config (или другом, выбранном разработчиками хранилище). Если проверка прошла успешно, сервер перенаправляет клиента к той странице, которую клиент запрашивал:

```
HTTP/1.1 302 Found
Location: /Default.aspx
Set-Cookie: AuthCookie=45F68E1F33159A9158etc.; path=/
htmlheadtitleObject moved/title/headbody  .
```

Заголовок Set-Cookie содержит аутентификатор клиента. Метод шифрования *cookie* (в данном примере 3DES), в случае ASP.NET задается в файле web.config.

- клиент повторяет запрос к ресурсу, но уже с только что установленным аутентификатором (*cookie*), который автоматически добавляется в заголовок HTTP:

```
GET /Default.aspx HTTP/1.0
Cookie: AuthCookie=45F68E1F33159A9158etc  .
```

- сервер проверяет *cookie*, и в случае успеха, предоставляет требуемую страницу с HTTP сообщением 200 OK.

Cookie может быть действительна только для текущей сессии браузера, но может быть создана и для длительного хранения. Каждый раз, когда пользователь запрашивает страницу с сервера, браузер автоматически отправляет *cookie* серверу. Сервер проверяет *cookie* по своей базе идентификаторов и, при наличии в базе такого идентификатора, разрешает пользователю доступ.

Файлы *cookie* – это текстовые файлы, содержащие пары имя/значение. Кроме того, они могут содержать информацию о сроке действия, пути и доменном имени (RFC2965). Браузер MS Internet Explorer хранит их в виде отдельных файлов в папке *Application Data\cookies.* Браузеры на основе Netscape и Mozilla хранят их в одном файле, называемом *cookies.txt*. А поскольку они сохраняются в файл, то могут быть изменены с помощью текстового редактора.\

Сессионные *cookie* подделать сложнее, однако существуют программные средства, например, *Burp Proxy*, которые позволяют просматривать и изменять сессионные *cookie* .

Аутентификация на основе форм уязвима к атакам *brute force*. Кроме того, это протокол уязвим к атакам прослушивания (*eavesdropping*) сети и атакам повтора сообщений (*replay attacks*), если не используется защита сообщений с помощью HTTPS или других криптопротоколов.

Если *cookie* содержат конфиденциальную информацию, для них могут быть установлены два атрибута: *secure* и *HTTPOnly*. Если установлен атрибут *secure,* браузер не станет передавать *cookie* вне HTTPS соединения. Атрибут *HTTPOnly* предолжен Microsoft и предназначен для защиты от атак перехвата сессии (*session hijacking*). Браузеры, поддерживающие этот атрибут, не позволяют программам на JavaScript получать доступ к *cookie,* даже если политика безопасности такой доступ разрешает.

Обзор атак на схемы аутентификации Web-приложений приведены в [2].

Заключение

Аутентификация является важнейшим механизмом безопасности Web-сервера. Поскольку требования к функционалу и безопасности Web-сервером могут существенно различаться, не существует одного, «наилучшего» способа аутентификации. Однако можно сформулировать основные требования к реализации данного механизма, реализация которых существенно снижает риск осуществления атак. К таким требованиям можно отнести:

- строгая политика управления паролями и учетными записями пользователей;
- невозможность обхода механизма аутентификации, например, с помощью инъекций;
- запрет использования персональных данных в качестве аутентификаторов;
- пересылаемые по сети аутентификаторы должны быть защищены с помощью протокола HTTPS;
- вся получаемая от пользователей информация должна тщательно контролироваться;
- идентификаторы сессии должны создаваться таким образом, чтобы их нельзя было предсказать.

При разработке подсистемы аутентификации необходимо определить требуемый уровень защищенности. Выбор уровня защищенности можно рассматривать как компромисс между удобством пользователя, производительностью и защищенностью. Необходимо учитывать все особенности используемых криптографических методов и протоколов. При реализации криптографических алгоритмов, таких как протокол формирования хеша SHA-1, протокол аутентификации сообщений HMAC, протоколы высокого уровня, например, SSL необходимо понимать круг задач, решаемых с их помощью.Рекомендации по построению схем аутентификации можно найти в прилагаемом списке литературы.

ЛИТЕРАТУРА

1. The WASC Threat Classification v2.0. [Электронный ресурс] - Режим доступа.

2. http://projects.webappsec.org/w/page/13246978/Threat-Classification свободный. — Загл. с экрана.

3. Authentication and Session Management on the Web [Электронный ресурс] - Режим доступа. http://www.westpoint.ltd.uk/advisories/Paul_Johnston_GSEC.pdf свободный. — Загл. с экрана.

4. Web Authentication Security [Электронный ресурс] - Режим доступа.

5. http://www.sans.org/reading_room/whitepapers/webservers/web-authentication-security_1250 свободный. — Загл. с экрана.

6. A Guide to Web Authentication Alternatives [Электронный ресурс] - Режим доступа: http://unixpapa.com/auth/ свободный. — Загл. с экрана.

7. Хогланд Г. Взлом программного обеспечения: анализ и использование кода [Текст]/ Хогланд Г., Мак-Гроу Г. // Издательский дом "Вильяме". - М.: -2005, 389 с.: ил. - ISBN 5-8459-0785-3, 0-201-78695-8.

8. CodeNet - Все для программиста! [Электронный ресурс]: Поиск уязвимостей в программах с помощью анализаторов кода. - Режим доступа: http://www.codenet.ru/progr/other/code-analysers.php свободный. — Загл. с экрана.

Надійшла: 24.07.2012 р.

Рецензент: д.т.н., професор Юдін О.К.